Computer Networks: Assignment 1

30 January 2017

Solving and submitting you assignment

Requirements about the delivery of this assignment:

- Submit via Blackboard (`http://blackboard.ru.nl`);
- Upload one single zip file containing two files, your solution of the assignment plus Python server program;
- The solution file as well as the archive file should take the name of your student number, for example student *s0123456* should submit a file named *s0123456.pdf* placed in the archive file *s0123456.zip*. Python server program should be a separate file in the archive named server.py.
- Write both your name and student number into the document (and only your student number in the filename).

Deadline: Wednesday, February 15, 20:00 sharp!

Goals: After completing these exercises successfully you should be able to:

- design and implement simple protocols;
- get familiarized with sockets and common network utilities;
- capture and interpret traces with Wireshark;
- run and understand traceroute output.

Marks: You will be graded with marks from 0 to 3 where 0 means not serious, 1 means serious but insufficient, 2 means sufficient and 3 means good. You can have at most 1 assignment graded 0. To get 1 or more, you MUST attempt to solve ALL exercises, even if the provided solution is not correct/complete. In other words, leaving an exercise out automatically turns your grade to 0. In your solution, please explain all answers clearly and concisely.

1 Wireshark to understand network utilities

a) Install the NetKit virtual environment at the link `http://wiki.netkit.org/index.php/Download_Official` (scroll down to *Live CD/DVD/USB* and download *Netkit live DVD/USB*) on a virtual machine. You can choose any of the well known Virtual Machine clients (VirtualBox and VMWare) to create the virtual machine. Within the virtual machine, open a terminal, and use the network utility *ifconfig*. Enumerate the network interfaces (left column) and their associated IPv4 addresses (*inet addr* field).

b) The network utilities *ping*, *traceroute* and *whois*[1] are among the most widely used in networking, and are available from the terminal. On the created Linux machine, run each of these commands on an address of your choice, capturing the traffic generated with Wireshark. (1) Describe the output briefly, (2) give print-outs of the captured traffic (apply a filter if necessary to weed out unwanted traffic) and (3) in a few sentences, give an interpretation to the captured traces.

For *ping* and *traceroute* use you can use any well known address (say *google.com*). For *whois*, use the IP *128.119.245.12*. For all commands, it is necessary and enough to supply the address as parameter.

2 Wireshark intro to HTTP

Open your favorite browser, clear its web cache and access `http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html` and capture the traffic generated using traceroute. You can use the Introductory and HTTP Wireshark labs as reference material, both of which come with the assignment.

a) Give a Wireshark filter that eliminates all non-HTTP packets as well as packets that don't originate from the IP of the server.

b) Print out (or copy paste) the stream of HTTP packets. To get the flow, right click on an HTTP packet, hover over *Follow* then click *HTTP Stream*.

3 The hopping problem

Use *traceroute*, on the address *soest.hawaii.edu*.

a) Give a print-out of the output.

b) What is the number of hops to the server at the given address? What is the average round trip time to the server?

c) Enumerate pairs of successive hops suggesting the route crossed an ocean. *(hint: Use* whois *for localizing the IPs of nodes, don't rely solely on RTTs, as you might get surprises.)*. Assume that UK is connected to Europe (thus routes to it from the Netherlands, do not hop over an ocean).

4 Protocol for a teller machine

Design and describe an application-level protocol to be used between an automatic teller machine (ATM) and a bank's centralized computer. Your protocol has two communicating entities: the ATM and the bank's centralize computer (or server).

[1] whois is available via apt-get

Your protocol should allow (1) a user's card and password to be verified; (2) the account balance to be queried; (3) and a withdrawal from the account to be made; (4) termination of a session. Your protocol should address the all-too-common case in which there is not enough money in the account to cover the withdrawal.

a) Specify your protocol by listing the types of messages from the ATM to the bank's server, and from the bank's server to the ATM. Each direction can imply different messages. For each message type, give a short description of the action defined. *(hint: a possible message type from the ATM to the server is PASSWD(userPWD). This message is sent to the server after the user has introduced a PIN, in order for the server to authenticate the user)*

b) Sketch the operation of your protocol for the case of a simple withdrawal with no errors, using a diagram similar to that in Figure 1.

c) Explicitly state the assumptions made by your protocol about the underlying end-to-end transport service.

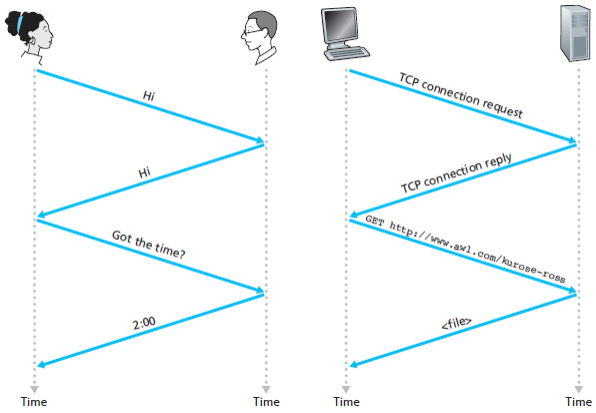

Figure 1: Human and machine protocols

5 Design and implement a transfer protocol

Assume a client-server application layer protocol for accessing file content on the server-side. The server is listening at port 20000, address localhost, for incoming clients. In the typical scenario, the client connects to the server, then sends a *hello* message. The server responds with its own *hello* message. Once the *hello* exchange is done, the client can access to files on the server side via the *get file_path* command, where *file_path* is a path relative to the server's local directory. If the file was found, the server responds to such queries with the message *file_content* (without transmitting the actual content). Otherwise, it responds with *file_not_found*, signaling that the file was not found. Once the client has retrieved all they desired, they can close the interaction using the *bye* message. The server responds with its own *bye* and closes the socket, and awaits connections from other clients. The server should also be open to the possibility that the client application crashes/unexpectedly closes socket at any point in the protocol, in which case, again, it should resume listening for other clients.

Any message of a different type to those already described is deemed intelligible by the server, which then sends *cannot_understand* message and ignores it. Similarly, any valid message that appears out of order /superfluous is also ignored. (say a second *hello* or a *bye* without having exchanged *hello's*)

a) Describe the server behavior of this protocol using a state machine representation. The inputs accepted are inputs/actions from the client, and outputs are server responses. The server might stay silent (no output) . You can assume the initial state of the server to be the state where it is waiting for clients. You should use the messages outlined, plus a few new others for common client actions (connect to server, close socket...). For the latter pick intuitive names. Figure 2 provides the state machine of a hypothetical echo server. You can use it as a guiding example.

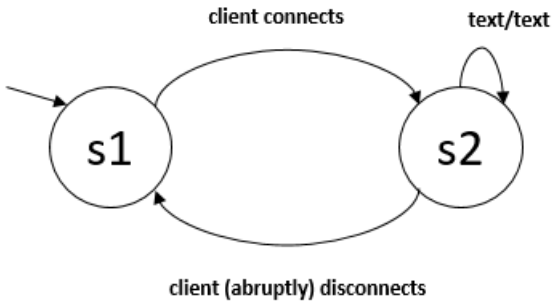

Figure 2: Echo Server State Machine. **s1** is the initial state (denoted by the pointing left arrow). When a client connects, the server transitions to **s2**. While here, it echoes any text inputted by the client until the client terminates connection, in which case the server transitions back to **s1**

b) Implement the server in Python 3.x using TCP sockets. Name it *server.py* and attach it with the assignment. You are provided code for an echo server with the assignment. You can adapt it to fit the requirements, or you can build a new server from scratch. You should use Chapter 2.7 in the book as reference material. The server need not handle concurrent requests. *(hint: use* telnet *to debug/test your server over localhost)*

c) Name a widely used application protocol this protocol is most closely related to. (in terms of the functionality offered) Explain the relation.